Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 33 (2006) 1713–1740
computers & operations research
www.elsevier.com/locate/cor

Ant colony optimization for the cell assignment problem in PCS networks

Shxyong Jian Shyu[a,*], B.M.T. Lin[b], Tsung-Shen Hsiao[a]

[a]*Department of Computer Science and Information Engineering, Ming Chuan University, Gwei-Shan, Tao-Yuan County, Taiwan 333, ROC*
[b]*Department of Information and Finance Management, Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan 300, ROC*

Available online 8 March 2005

Abstract

Even though significant improvement to communications infrastructure has been attained in the personal communication service industry, the issues concerning the assignment of cells to switches in order to minimize the cabling and handoff costs in a reasonable time remain challenging and need to be solved. In this paper, we propose an algorithm based upon the Ant Colony Optimization (ACO) approach to solve the cell assignment problem, which is known to be $\mathcal{NP}$-hard. ACO is a metaheuristic inspired by the foraging behavior of ant colonies. We model the cell assignment problem as a form of matching problem in a weighted directed bipartite graph so that our artificial ants can construct paths that correspond to feasible solutions on the graph. We explore and analyze the behavior of the ants by examining the computational results of our ACO algorithm under different parameter settings. The performances of the ACO algorithm and several heuristics and metaheuristics known in the literature are also empirically studied. Experimental results demonstrate that the proposed ACO algorithm is an effective and competitive approach in composing fairly satisfactory results with respect to solution quality and execution time for the cell assignment problem as compared with most existing heuristics or metaheuristics.
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Cell assignment; Ant colony optimization; Metaheuristic; Multi-agent

* Corresponding author. Tel.: +886 3 3507001x3402; fax: +886 3 3593874.
E-mail address: sjshyu@mcu.edu.tw (S.J. Shyu).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2004.11.026

1. Introduction

Since the last decade, there have been significant advances in the development of mobile communication systems. Mobile networks in the next few years could be efficiently migrated to broadband services based on high-speed wireless access technologies [1]. The backbone networks that are fostering current research include the public land mobile networks (PLMN), mobile Internet protocol networks, wireless asynchronous transfer mode (WATM) networks, and low earth orbit satellite networks [2]. Even though significant improvement to communications infrastructure has been attained in the personal communication service industry, the issues concerning the assignment of cells to switches in order to minimize the cabling and handoff costs in a reasonable time remain challenging and need to be resolved. In this paper, we address one of the critical problems concerning how to assign cells to switches in order to minimize the cost that is usually considered by the designers of such mobile communication services or personal communication services (PCS).

In PCS networks, each cell has an antenna that is used to communicate with subscribers over some pre-assigned radio frequencies. Groups of cells are connected to a switch, through which the cells are then routed to the terrestrial (PLMN, ATM, Internet) or satellite networks. Fig. 1 shows an example where cells A and B are connected to switch s_1 while cells C and D are connected to switch s_2. Suppose that a subscriber is currently talking to someone and this call is transmitted through cell B and switch s_1. If the subscriber moves from cell B to A, switch s_1 will perform a *handoff* for this call. This call does not trigger any location update in the database that records the position of the subscriber. Besides, the handoff does not entail any network entity other than switch s_1. Suppose that the subscriber moves from cell B to C. Then the handoff involves not only the modification of the location of the subscriber in the database but also the execution of a fairly complicated protocol between switches s_1 and s_2. Therefore, there are two types of handoffs, one involves only one switch and the other involves two switches. The latter is relatively more difficult and costly than the former. For a more comprehensive description of handoffs, the reader is referred to Yacoub [3] and Merchant and Sengupta [4].

Obviously, the cells among which the handoff frequency is high should be assigned to the same switch as far as possible to reduce the cost of handoffs. However, since the call handling capacity of each switch is limited, we should take into account this constraint. Incorporating the *cabling cost* that occurs when a call is connected between a cell and a switch, we have an optimization problem, called the *cell assignment problem* [4], of assigning cells to switches such that the total hybrid cost, comprising handoff cost between adjacent cells and cabling cost between cells and switches, is minimized under the constraints of the call handling capacities of switches.

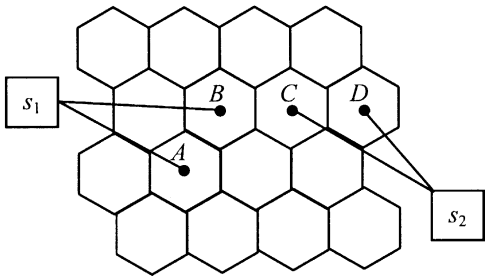

Fig. 1. Cells are assigned to switches.

This problem is a generalized version of the bin-packing problem which is known to be $\mathcal{NP}$-hard [4,5]. Therefore, Merchant and Sengupta [4] proposed a heuristic based upon the greedy strategy to find approximate solutions to this problem. Bhattacharjee et al. [6] proposed more heuristics for this problem and compared their performances in terms of solution quality and execution time. Approaches based upon metaheuristics for solving the cell assignment problem can also be found in the literature, including *simulated annealing* [7], *tabu search* [8,9] or *genetic* and *memetic* algorithms [10].

In this paper, we will solve the cell assignment problem by applying the Ant Colony Optimization (ACO) approach to explore the possibility of composing better solutions. ACO utilizes a nature metaphor originating from the food seeking behavior of ant colonies. It has been successfully applied to cope with many classical optimization problems. In general, metaheuristics might get solutions with a better solution quality than simple heuristics at the expense of more computation time. Thus we expect that our ACO algorithm can get satisfactory solutions in a reasonable time. Such a temporal requirement is critical for practical applications, especially in a dynamically changing environment where the cell-to-switch assignment should be updated in a timely manner. Therefore the goals in this paper are: (1) to design an ACO algorithm to deal with the cell assignment problem; (2) to conduct experiments to investigate its behavior in the optimization process; and (3) to empirically study the effectiveness and efficiency of our ACO algorithm and the heuristics and metaheuristics mentioned above.

The rest of the paper is organized as follows. The mathematical model of the cell assignment problem and the heuristics or metaheuristics already proposed in the literature for this problem are described in Section 2. Section 3 presents the fundamental concept and structure of ACO. In Section 4, we shall propose the specific features of the ACO algorithm for solving the cell assignment problem. Our preliminary experiments for composing a satisfactory parameter setting for ACO and the computational comparison among ACO and other approaches are presented in Section 5. Finally, we give some concluding remarks in Section 6.

2. Mathematical model and heuristic approaches

2.1. Mathematical model

The cell assignment problem was mathematically formulated by Merchant and Sengupta [4] as follows. Assume n cells are to be assigned to m switches. The locations of the cells and switches are fixed and known. If cells i and j are assigned to different switches, then a cost is incurred every time a handoff occurs between cells i and j. Let h_{ij} be the cost per time unit for the handoffs that occur between cells i and j, $1 \leqslant i,\ j \leqslant n$. Usually, h_{ij} is proportional to the frequency of handoffs that occur between cells i and j per time unit. The frequency is assumed as known in advance. Let c_{ik} be the amortized cabling cost per time unit between cell i and switch k, $1 \leqslant i \leqslant n$ and $1 \leqslant k \leqslant m$. Let λ_i denote the number of calls that cell i handles per time unit and M_k denote the call handling capacity of switch k. Our objective is to assign each cell to a switch so as to minimize the total cost per time unit. The total cost per time unit consists of two components: the handoff cost for handoffs between cells associated with different switches and the cabling cost between the cells and the switches. The optimization issue has to be carried out in such a way that the call handling capacity of each switch is not violated.

To formulate this problem into an integer programming model, Merchant and Sengupta [4] first introduce binary decision variables defined as

$$x_{ik} = \begin{cases} 1, & \text{if cell } i \text{ is assigned to switch } k; \\ 0, & \text{otherwise.} \end{cases}$$

Since each cell must be assigned to exactly one switch, we have

$$\sum_{k=1}^{m} x_{ik} = 1, \quad 1 \leqslant i \leqslant n. \tag{1}$$

The capacity constraint of a switch is given by

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leqslant M_k, \quad 1 \leqslant k \leqslant m. \tag{2}$$

The handoff cost between cell i and j is determined by whether both cells i and j are connected to a common switch k. The situations are defined as

$$z_{ijk} = x_{ik} x_{jk}, \quad 1 \leqslant i, j \leqslant n \quad \text{and} \quad 1 \leqslant k \leqslant m \tag{3}$$

and

$$y_{ij} = \sum_{k=1}^{m} z_{ijk}, \quad 1 \leqslant i, j \leqslant n. \tag{4}$$

That is, $y_{ij} = 1$ if cells i and j are connected to some common switch k; otherwise it is 0. Consequently, the total handoff cost per time unit among cells is given by

$$\sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij}(1 - y_{ij}).$$

The total cabling cost is

$$\sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik} x_{ik}.$$

Therefore, the objective function of the cell assignment problem is to minimize

$$\sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik} x_{ik} + \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij}(1 - y_{ij}). \tag{5}$$

Combining formulae (1)–(5) and other constraints on the decision variables, Merchant and Sengupta [4] successfully formulate the cell assignment problem as an integer programming problem. Some heuristic or metaheuristic approaches to solving this $\mathcal{NP}$-hard problem are introduced in the next Sub-section.

2.2. Heuristic and metaheuristic approaches

Heuristic and metaheuristic approaches to solving the cell assignment problem have been proposed in the literature since the middle 1990s. In this section, we describe some existing approaches. Merchant and Sengupta [4] proposed a heuristic based upon a greedy strategy (denoted as *H*). The heuristic iteratively finds $b(>1)$ minimum assignments for the jth, $1 \leqslant j \leqslant n$, cell-to-switch assignment that keeps the previous $j-1$ assignments (containing b candidates also) unaltered. As a result, it could produce a feasible solution which might be better than that produced by simply applying a greedy method with $b = 1$. The heuristic further performs pair-wise exchanges on the cell-to-switch assignments in this feasible solution if a reduction of cost is attainable. Numerical results of their experiments show that for the instances where the number of cells is less or equal to 30, this heuristic can come up with solutions that are close to the optimum with an average relative error of 2%.

More heuristics and experiments could be found in Bhattacharjee et al. [6]. In the following, we briefly introduce two of the most effective heuristics in their study, Heuristic II (*H*-II) and Heuristic IV (*H*-IV). The idea of both heuristics arises from the concept about clustering: the cells among which the handoff frequency is high should be grouped as a cluster (i.e., assigned to a switch) to possibly minimize the handoff cost. Initially, every cell per se is a cluster. The clusters grow (or merge) by absorbing its neighbor cells until the number of clusters is equal to the number of switches. The cumulative sum of handoff costs of a cell with all its neighbors already assigned to some cluster is considered in *H*-II to determine which switch the cell is to be assigned to without violating the capacity constraint.

In *H*-IV, the consideration is based on the average handoff cost of a cell, i.e., the ratio of handoff cost to cabling cost. Therefore, a switch is represented as a cluster, and the cells attached to the switch become the seeds of the cluster. All clusters simultaneously pick up their respective neighbor cells to expand their coverage in an optimal way. Here the optimal way means to achieve an assignment with a minimum handoff volume. That is, if a cell has a high handoff volume with one cluster, then it is potentially beneficial to assign it into the cluster to reduce the incurred handoff cost. Bhattacharjee et al. [6] designed computational experiments and the numerical results show that no single heuristic can performs well in both aspects of cost and execution time. However, heuristics *H*-II or *H*-IV outperform other heuristics in most of their test cases where the number of cells ranges from 25 to 484.

Approaches based upon metaheuristics for the cell assignment problem include simulated annealing (SA) [7,11] tabu search (TS) [8,9], and genetic and memetic algorithms [10]. It is beyond the scope of this paper to go into details of these metaheuristics. The readers may refer to the above-mentioned papers. We only point out that it is reported in [8,9] that *TS* outperforms *SA* when the number of cells ranges from 15 to 200 and the number of switches is between 2 and 7. Also, when the number of cells is generated from the same range and the number of switches is between 5 and 7, the computational results in [10] reveal that memetic algorithm (MA), a population-based evolutionary algorithm incorporating a local search strategy (using *SA* or *TS*), is superior to the standard genetic algorithm for both of the solution quality and execution time; further, *MA* is slightly better than *TS* and *SA* in delivering more satisfactory solutions at the expense of more computation time.

3. Fundamentals and applications of ACO

Ant algorithms were first proposed by Dorigo and his colleagues [12,13] as a multi-agent approach to solving difficult combinatorial optimization problems. ACO was inspired by some observations on real

```
procedure ACO_metaheuristic()
    initialize_pheromone_and_ant_memory();
    while (termination criterion not satisfied) do
        for (each ant in the colony)
            M = update_ant_memory();
            while (current_state ≠ target_state)
                A = read_local_ant-routing_table();
                P = compute_transition_probabilities(A, M, problem_constraints);
                next_state = apply_ant_decision_policy(P, problem_constraints);
                local_pheromone_update();
                update_ant-routing_table();
                M = update_internal_state();
            endwhile
        endfor
        pheromone_evaporation();
        global_ pheromone_update();
        update_ant-routing_table();
        daemon_actions();          {optional}
    end while
end procedure
```

Fig. 2. Outline of the ACO metaheuristic.

ant colonies. While traveling from their colony to the food source back and forth, ants deposit pheromone on the ground forming a pheromone trail. Ants can sense pheromone. When deciding a path to follow, they tend to choose the ones with strong pheromone intensities. The pheromone trail guides the ants to find their way back to the nest or to the food source. In this way, shorter paths would accumulate more pheromone than longer ones. Thus the pheromone-trail-following behavior forms a positive feedback mechanism such that a colony of ants may be able to exploit the pheromone trails formed by the individual ants to discover the shortest path between the nest and the food source.

ACO has been successfully applied to resolve the traveling salesman problem (TSP) [12,14,15], graph coloring problem [16], quadratic assignment problem [17], generalized minimum spanning tree (GMST) problem [18], sequential ordering problem [19], vehicle routing problem [20–22] and scheduling problems [23–26]. Some researchers presented successful ACO applications in the area of telecommunication networks [27–30]. Due to the attractiveness about the behavior of ants and the simplicity in implementations, ACO has been gaining more and more research attention since the early 1990s.

To design effective ACO algorithms, we first transform the original problem into a certain structure, say a graph, which is suitable for artificial ants (agents) to traverse and update pheromone trails. Second, we need to design a heuristic, usually a greedy one, to guide the search for solutions in the early stage of the computational process. Third, we devise an autocatalytic feedback process by managing the pheromone trails in order to acquire knowledge or experience from synergetic ants. Meanwhile, the constraint satisfaction method should be established to rule out infeasible solutions. Then, the probabilistic interaction among the agents mediated by the pheromone trails could produce good solutions to the problem under study. Fig. 2 presents the outline of the ACO metaheuristic, a modified version from [31].

Suppose that the problem has been transformed into a structure suitable for ants to traverse. Each ant in the colony has a local memory to memorize its traversal information. A functional composition of the locally available pheromone and heuristic values defines the *ant-routing table* for a certain ant. Each ant moves through adjacent states of the problem by applying a *stochastic local decision policy*, the *state transition rule*, which depends upon the ant's routing table, local memory and the problem constraints. By moving from state to state, ants incrementally construct solutions to the original problem in a probabilistic way. Ants can build an autocatalytic feedback process for the experiences about their search for good solutions by adjusting the intensities of pheromone trails. An ant might perform a *local pheromone update* to change the pheromone of the just visited arc to favor the unvisited arcs whenever it moves to the next state. After all ants in the colony have constructed their own solutions, the pheromone intensity would be decreased due to the *evaporation* effect to reflect the natural phenomenon so as to forget bad decisions ever made. The global pheromone update mechanism will be triggered to add new pheromone onto the path that corresponds to a solution that is better than others in the colony. Whenever the pheromone trails are changed, the ant-routing table should be updated accordingly. The optional *daemon_actions*() can be employed to implement the centralized actions such as depositing additional pheromone to reward some best solution found thus far from a non-local perspective, or carrying out a *local search* procedure to improve the quality of the solution, and so on.

The design details of the ACO algorithm will be explained in the next section.

4. Applying ACO to cell assignment problem

The problem transformation, transition probabilities, pheromone updating rules, local search, and stopping criterion of the ACO algorithm for the cell assignment problem will be presented in the following sub-sections respectively.

4.1. Problem transformation

First of all, we seek to devise a representation structure that is suitable for ants to search for solutions to the problem. In the literature, graphs have been widely adopted as such intrinsic structures. To name a few as examples, Dorigo et al. [14] and Dorigo and Gambardella [15] used a weighted directed graph for TSP where a *cycle* in the graph corresponds to a feasible solution to TSP; Shyu et al. [18] modeled the GMST problem as a weighted undirected graph such that a *tree* could be a feasible solution to GMST. In this paper, we shall model the cell assignment problem by a *weighted directed bipartite graph* where a *path* represents a feasible solution as described in the following. Note that a bipartite graph has a set of vertices decomposed into two disjoint sets such that no two vertices within the same set are adjacent.

Consider a cell assignment problem instance P with n cells and m switches. Let C denote the set of the n cells and S the set of the m switches in P. We may use a directed bipartite graph $G=(V, E)$ with weights associated with both vertices and directed edges (or arcs) to represent P, where $V = C \cup S$, $C \cap S = \emptyset$, the weight λ_i on vertex $i \in C$ denotes the call volume of cell i, the weight M_k on vertex $k \in S$ denotes the capacity of switch k and the weight c_{ik} on arc $\langle i, k\rangle \in E$ ($i \in C$ and $k \in S$) denotes the cabling cost between cell i and switch k. As a way, a feasible assignment corresponds to a directed bipartite sub-graph $G' = (C \cup S, E') \subseteq G$, where $E' = \{\langle i, k\rangle \mid i \in C \text{ and } k \in S\} \subseteq E$ with the constraints: (1) $|E'| = n$ and if arcs $\langle a, b\rangle, \langle c, d\rangle \in E'$, then $a \neq c$, that is, every cell should be assigned to exactly one switch; and

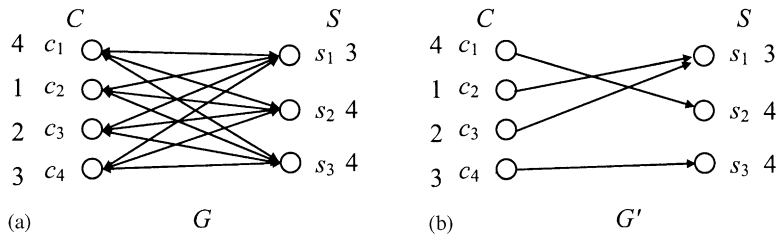

Fig. 3. Directed bipartite graph representation for the cell assignment problem (a) G and (b) G'.

(2) $\sum_{\langle i,k\rangle \in E'} c_{ik} < M_k$, $1 \leqslant k \leqslant m$, the constraint on each switch's capacity should be satisfied. Note that the existence of arc $\langle i, k\rangle \in E'$ in G' indicates the assignment of cell i to switch k.

Fig. 3(a) illustrates an instance P with $C = \{c_1, c_2, c_3, c_4\}$ and $S = \{s_1, s_2, s_3\}$. Instance P is represented by a weighted directed bipartite graph $G = (C \cup S, E)$, where every vertex in C has three arcs pointing to the three switch, every switch in S has four arcs pointing to the four vertices, and the weights (call volumes for cells and capacities for switches) are annotated beside the corresponding vertices. Fig. 3(b) shows a possible feasible assignment of P, where $E' = \{\langle c_1, s_2\rangle, \langle c_2, s_1\rangle, \langle c_3, s_1\rangle, \langle c_4, s_3\rangle\}$.

Although G' describes well a possible feasible assignment of P, the arcs in G' are not connected to form a path. Thus it is not easy for our artificial ants to traverse the arcs in G. We might rearrange the order of the arcs in E', say $E' = \{\langle c_{a_1}, s_{b_1}\rangle, \langle c_{a_2}, s_{b_2}\rangle, \langle c_{a_3}, s_{b_3}\rangle, \ldots, \langle c_{a_n}, s_{b_n}\rangle\}$ where $1 \leqslant a_i \leqslant n$ and $1 \leqslant b_i \leqslant m$ for $1 \leqslant i \leqslant n$, and add $E'' = \{\langle s_{b_1}, c_{a_2}\rangle, \langle s_{b_2}, c_{a_3}\rangle, \ldots, \langle s_{b_{n-1}}, c_{a_n}\rangle\}$ into G' where $E'' \subseteq E$ and $|E''| = n - 1$ in such a way that the directed path composed of the $2n - 1$ arcs of E' and E'', namely $\langle c_{a_1}, s_{b_1}\rangle, \langle s_{b_1}, c_{a_2}\rangle, \langle c_{a_2}, s_{b_2}\rangle, \langle s_{b_2}, c_{a_3}\rangle, \ldots, \langle s_{b_{n-1}}, c_{a_n}\rangle, \langle c_{a_n}, s_{b_n}\rangle$, not only holds the information of each feasible cell-to-switch assignment but also forms a continuous directed path that is an ideal structure for our artificial ants to traverse in G. Along such a path, the movement of an ant from cell a_r to switch b_r corresponds to the assignment of cell a_r to switch b_r, and the subsequent movement from switch b_r to cell a_{r+1} corresponds to the decision about choosing cell a_{r+1} as the next cell to go on the following cell-to-switch assignment. Therefore by making n decisions on the two consecutive movements from a cell to a switch and then from that switch to another cell, each ant can construct a certain feasible solution to the cell assignment problem if all movements do not violate the problem constraints. Note that the last decision involves only one cell-to-switch decision. As a result, such a directed path, or a subgraph $G'' = (C \cup S, E' \cup E'')$ of G, where $E', E'' \subseteq E$ defined as above would be a appropriate representation for an feasible assignment that our ants can traverse it out of G.

To minimize the total cost, the best path shall contain the feasible assignment that minimizes

$$\sum_{i=1}^{n} c_{a_i b_i} + \sum_{i=1}^{n} \sum_{j=1}^{n} h_{a_i a_j} f_{a_i a_j}, \quad 1 \leqslant i, j \leqslant n, \tag{6}$$

where $f_{a_i a_j}$ determines whether cell a_i and cell a_j are assigned to the same switch:

$$f_{a_i a_j} = \begin{cases} 0, & \text{if } b_i = b_j; \\ 1, & \text{otherwise.} \end{cases}$$

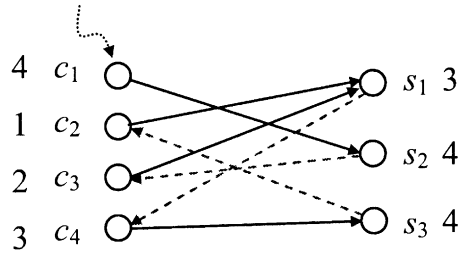

Fig. 4. Directed path in G corresponding to a solution to P.

Fig. 4 depicts the graph G'' corresponding to the feasible assignment in Fig. 3(b), where $E' = \{\langle c_1, s_2\rangle, \langle c_3, s_1\rangle, \langle c_4, s_3\rangle, \langle c_2, s_1\rangle\}$, $E'' = \{\langle s_2, c_3\rangle, \langle s_1, c_4\rangle, \langle s_3, c_2\rangle\}$ and the associated path is $\langle c_1, s_2\rangle, \langle s_2, c_3\rangle, \langle c_3, s_1\rangle, \langle s_1, c_4\rangle, \langle c_4, s_3\rangle, \langle s_3, c_2\rangle, \langle c_2, s_1\rangle$.

Since each cell should be assigned to some switch exactly once, we provide a short-term memory, called *tabu* (not the tabu list in the tabu search), for our artificial ants to memorize the cells that have been already assigned. Meanwhile, multiple cells might be assigned to a switch if its capacity constraint is not violated. Therefore, we associate each ant with another short-term memory, called *feas*, to keep track of the availability of switches. To find solutions to the cell assignment problem, an ant, say u, at cell i confronts two successive decisions to make: (1) which switch to move on (that is, which switch $k \in feas^u$ cell i will be assigned to), and (2) which unvisited cell to move on from the selected switch (namely, which cell $j \notin tabu^u$ to continue). We call it a *step* for an ant to make these two decisions in sequence. After n steps with the exception that the nth step needs only decision 1, the ant could construct a directed path, which corresponds to a feasible assignment. Such an n-step process is called an *iteration*. Consider the instance in Fig. 4. The solid arcs are the outcome of decision 1 while the dashed arcs are the outcome of decision 2. These arcs together constitute a directed path in G as shown in Fig. 4. At this stage, we have transformed the cell assignment problem to a weighted directed bipartite graph that facilitates the ants to construct solutions by traversing a path with a sequence of $2n - 1$ arcs alternatively from some vertex representing a cell to a certain vertex representing a switch and then from this switch to another vertex representing some not-yet-assigned cell.

4.2. State transition rules

Essentially, an ant decides its movements stochastically relying on the information elicited from the *pheromone intensity* and the *heuristic function*. Since an ant has to make two successive decisions in each step and these two decisions refer to different knowledge with respect to the problem status, (i.e., decision 1 is about choosing a switch with respect to the capacity constraint while decision 2 is about selecting a cell with respect to its call volume,) we employ two different greedy heuristic functions to reflect the different preferences for these two decisions, respectively. Also we utilize pheromone trails to keep track of the ants' experience in searching for good solutions on the visited arcs in G. Suppose that ant u is currently positioned at cell i. The state transition rules for the two decisions are:

4.2.1. Choosing an arc from cell i to some switch k

Suppose that at step r, ant u chooses arc $\langle i, k\rangle$ to move on from cell i (also denoted as a_r) to switch k (also denoted as b_r), $1 \leqslant r \leqslant n$. A partial cost incurred during these r steps can be computed and employed

as the basis of our heuristic. Let $g_{a_r b_r}$ denote the partial cost incurred during these r steps. The partial cost $g_{a_r b_r}$ can be calculated by

$$g_{a_r b_r} = \begin{cases} c_{a_1 b_1} & \text{if } r = 1; \\ g_{a_{r-1} b_{r-1}} + c_{a_r b_r} + \sum_{l=1}^{r-1} h_{a_r a_l} f_{a_r a_l} & \text{if } r > 1 \text{ and } f_{a_r a_l} = \begin{cases} 0, & \text{if } b_r = b_l; \\ 1, & \text{otherwise,} \end{cases} \end{cases} \tag{7}$$

where the value of $f_{a_r a_l}$ depends on whether cell a_l (the cell visited by ant u at step l, $1 \leqslant l < r$) and cell a_r (the cell on which ant u is) are assigned to the same switch.

We use the instance in Fig. 4 as an example to explain how to calculate the partial cost in decision 1. At the first step, the ant moves from cell 1 to switch 2. Thus $g_{12} = c_{12}$. At the second step, the ant moves from cell 3 to switch 1. We have $g_{31} = g_{12} + c_{31} + h_{31} f_{31}$. Note that $f_{31} = 1$, because cells 3 and 1 are not assigned to the same switch. After step three, $g_{43} = g_{31} + c_{43} + h_{41} f_{41} + h_{43} f_{43}$, where $f_{41} = 1$ and $f_{43} = 1$. At the end of step four, $g_{21} = g_{43} + c_{21} + h_{21} f_{21} + h_{23} f_{23} + h_{24} f_{24}$, where $f_{21} = 1$, $f_{23} = 0$ (because cells 2 and 3 are both assigned to switch 1) and $f_{24} = 1$.

We use the inverse value of the partial cost $g_{a_r b_r}$ as our heuristic function $\eta_{a_r b_r}$ at step r of some iteration. Then, the probability that ant u positioned at cell i (a_r) moves to switch k (b_r) at the rth step is defined as follows:

$$p_{ik} = \begin{cases} 1 & \text{if } q \leqslant q_0 \text{ and } k = \arg \max_{s \in feas^u} \{\tau_{is} \eta_{is}^{\beta}\} \quad \text{(exploitation)}, \\ \dfrac{\tau_{ik} \eta_{ik}^{\beta}}{\sum_{s \in feas^u} \tau_{is} \eta_{is}^{\beta}} & \text{if } q > q_0 \text{ and } k \in feas^u \quad \text{(biased exploration)}, \\ 0 & \text{otherwise } (k \notin feas^u) \quad \text{(infeasible)}, \end{cases} \tag{8}$$

where τ_{ik} denotes the intensity of pheromone on arc $\langle i, k \rangle$ referred in decision 1, $i \in C$ and $k \in S$, $q_0 (0 < q_0 \leqslant 1)$ is the relative preference for exploitation or exploration, $\beta (\beta \geqslant 0)$ reflects the relative importance between pheromone intensity and heuristic value, and $feas^u$ denotes the feasible set of switches that the cells can be assigned to without violating the capacity constraints for ant u. Ant u at cell i picks up a pseudo random number $q (0 \leqslant q \leqslant 1)$ to make its movement at step r: if $q \leqslant q_0$, it moves to switch k through arc $\langle i, k \rangle$ that achieves the maximum value of $\tau_{ik} \eta_{ik}^{\beta}$; if $q > q_0$, it moves to switch k with a biased probability in order to avoid *stagnation* by exploring other areas in the solution space for better solutions. It is easy to see that we prefer a switch k from cell i (an arc $\langle i, k \rangle$) with a higher pheromone intensity (more ant knowledge) and a higher heuristic value (lower partial cost by our greedy approach) for decision 1.

Tuning the values of q_0 allows us to have an adaptive degree between exploitation and biased exploration, i.e., to exploit the steps that seem to be the most promising or to probabilistically explore the search space. The pheromone intensity τ_{ik} on arc $\langle i, k \rangle$ would be decreased according to the local pheromone update rule defined in the following (Section 4.3) to diversify the search. Once ant u has made the decision that cell i be assigned to switch k (switch k would handle the calls from cell i), the capacity M_k of switch k should be subtracted by λ_i. If M_k is no more available for any of the un-assigned cells, switch k becomes unavailable and it should be removed from $feas^u$. At the end of each iteration, $feas^u$ is reset for the next ant.

4.2.2. Following an arc from switch k to an un-visited cell j

We employ another greedy heuristic for ants to choose a cell not yet assigned to move on for decision 2. The not-yet-assigned cell with the heaviest call volume is considered first in each step. Since ant u memorizes the cells that have been traversed in the current iteration in $tabu^u$, the heuristic function η'_j is defined as the call volume of cell j, which does not belong to $tabu^u$ for ant u, i.e., $\eta'_j = \lambda_j$ if $j \notin tabu^u$; 0 otherwise. We use the following probability function for ant u at switch k to make its choice to move.

$$p'_{kj} = \begin{cases} 1 & \text{if } q < q_0 \text{ and } j = \arg \max_{s \notin tabu^u} \{\tau_{ks} {\eta'_s}^{\beta'}\} \quad \text{(exploitation)}, \\ \dfrac{\tau_{kj} {\eta'_j}^{\beta'}}{\sum_{s \notin tabu^u} \tau_{ks} {\eta'_s}^{\beta'}} & \text{if } q \geqslant q_0 \text{ and } j \notin tabu^u \quad \text{(biased exploration)}, \\ 0 & \text{otherwise } (j \in tabu^u) \quad \text{(tabu)}, \end{cases} \tag{9}$$

where τ_{kj} denotes the intensity of pheromone on arc $\langle k, j\rangle$ referred in decision 2, $k \in S$ and $i \in C$, $q_0 (0 < q_0 \leqslant 1)$ is the relative preference for exploitation or exploration, $\beta' (\beta'' \geqslant 0)$ is the relative importance of the pheromone intensity and heuristic value, and $tabu^u$ denotes the set of cells that have been visited by ant u. As can be seen that we favor a cell with a higher call volume and a viable arc with a higher pheromone intensity, $tabu^u$ should include cell j after it is chosen. At the end of each iteration, $tabu^u$ would be reset.

4.3. Pheromone updating rules

Each ant deposits or removes some amount of pheromone on the visited arcs. This mechanism provides a way of the *indirect communications* to share the knowledge about searching for good solutions amongst the colony. When and how much do the ants deposit or remove the pheromone are essential issues in the design of a good ACO algorithm. We prefer the arcs that constitute the minimum cost path derived thus far. Therefore at the end of each iteration, we undertake a *global pheromone update* on the arcs of the best path derived in the most recent iteration. For simplicity, we call such a path the *iteration-best* path. Let T^+ denote the iteration-best path and Q^+ denote the cost of T^+. The global pheromone updating rule is defined as follows:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\, \Delta\tau_{ij}, \quad \Delta\tau_{ij} = \begin{cases} n(Q^+)^{-1} & \text{if } \langle i, j\rangle \in T^+; \\ 0 & \text{otherwise}, \end{cases} \tag{10}$$

where $\rho (0 < \rho \leqslant 1)$ is the pheromone evaporation coefficient. What we intend to do is to take into account the tentative superiority of T^+ by adding pheromone intensity by $\rho\, \Delta\tau$, which is proportional to the quality of Q^+, onto the arcs belonging to T^+, while at the same time the pheromone intensity on all arcs in E is decreased according to the evaporation rate ρ to imitate the natural phenomenon evaporation over time in order to let the ants forget inferior decisions made in the earlier stages of the search process.

To keep away from early convergence (also called stagnation), a situation in which all the ants reconstruct the same solution and stop exploring new possibilities before some satisfactory solution is found, we perform *a local* (*step-by-step*) *pheromone update*. The local pheromone updating rule is defined as:

$$\tau_{ij} = (1 - \varphi)\tau_{ij} + \varphi\tau_0, \quad \text{if } \langle i, j\rangle \text{ has been selected by some ant}, \tag{11}$$

where $\varphi(0 < \varphi \leqslant 1)$ is the parameter controlling the degree of pheromone *decay*. The value of τ_0 is set to be the same as the initial value of the pheromone trails. Whenever an ant moves from a cell to a switch (or from a switch to a cell), the application of the local updating rule makes pheromone intensity on the corresponding arc decrease. This mechanism is designed to make the visited arcs less attractive as they are just visited by ants, and indirectly favor the exploration of unvisited arcs. As a consequence, ants tend not to converge to a common path in the earlier iterations. Consider the case that ant u_1 moves from cell i to switch k. Without pheromone decay, ant u_2 right after u_1 at cell i has a high probability to follow u_1 by choosing $\langle i, k\rangle$ to move on (likewise for the subsequent movement along $\langle k, j\rangle$). Thus, ants would be easily led to a stagnation situation. The intensity-decreasing effect introduced by the local pheromone update reduces the probability for early convergence to occur.

4.4. Local search

Whenever some ant finds an improved global-best solution at the end of some iteration, we employ a local search to improve this new so far global-best solution until some local optimum is reached. Our local search is a greedy approach. Suppose that a new global-best assignment is derived, i.e., $E' = \{(c_{a_1}, s_{b_1}), (c_{a_2}, s_{b_2}), \ldots, (c_{a_n}, s_{b_n})\}$ where $1 \leqslant a_i \leqslant n$ and $1 \leqslant b_i \leqslant m$ for $1 \leqslant i \leqslant n$. We would try to change the current assignment of each cell at most once as long as the re-assignment leads to a better solution. For each cell that has not been re-assigned yet, say cell a_i, we find the feasible re-assignment of cell a_i to switch $b_j, b_j \neq b_i$, which reduces the objective function by the largest amount. We then change the assignment of cell a_i from switch b_i to switch b_j and mark cell a_i as re-assigned. We continue this re-assigning process for those not yet re-assigned cells till the objective function cannot be further reduced or all cells have been re-assigned once.

4.5. Stopping criterion

The stopping criterion of ACO could be specified by a maximum number of iterations, a specified CPU time limit, or a given number of consecutive iterations within which no improvement on solutions is attained. Unless otherwise specified, in this paper, we set our criterion to the last one, which dictates the situation where further improvement is becoming less likely.

5. Experimental results and analysis

To test the effectiveness of our ACO algorithm (*ACO*) for the cell assignment problem, we design and conduct a series of computational experiments in this section. The default values of the parameters in *ACO*, except when discussing their individual settings, are set as: $\beta = \beta' = 2$, $q_0 = 0.1$, $\varphi = 0.05$, $\rho = 0.05$, $u = 16$ and $\tau = n/100$. These values were determined by extensive preliminary experiments. We also found that these tentative parameter values are mutually independent in most of the test instances in our experiments. The execution of *ACO* stops if no improvement on solutions can be found within 100 consecutive iterations. All of the programs are coded by Borland C++ Builder 5 and executed on a personal computer running MS-Windows 2000 with an AMD Athlon 1700+ CPU and 256 MB RAM.

Regarding the test problems, we assume that the cells lie on a hexagonal grid of roughly equal dimensions in two axes (see Fig. 1), and the antenna for each cell is at the center of the cell. Switches are uniformly distributed over all the cells, and are assumed to be at the center of the cell. In Section 5.1, based

upon the data sets generated from a simple uniform distribution, we present the results of our preliminary experiments for composing a satisfactory parameter setting for ACO. Also, from these experiments we investigate the detailed characteristics of the solution-finding sessions of the artificial ants. In Section 5.2, based upon the test case generation scheme adopted by Merchant and Sengupta [4], we further implement some heuristics and metaheuristics proposed in the literature and compare the performance of ACO with those of these existing approaches.

5.1. Parameters setting

First of all, we test *ACO* by using different parameters to determine the most advantageous setting as well as to justify the performance of the artificial ants in searching for good solutions. The test data generated in this section are simply based upon a uniform distribution. Call volume of cell i per time unit, λ_i, is uniformly distributed within interval [0.1, 8.0]. Cabling cost between cell i and switch k, c_{ik}, is proportional to their geometric distance. A call in cell i, which has t neighbors: cells $i_1, i_2, \ldots, i_t$, is uniformly generated according to λ_i. Further, with an equal probability, $1/(t+1)$, it is ended or transferred to one of cell i's neighbors. Handoff cost h_{ij} is proportional to the sum of λ_i and λ_j. The capacity of switch k, M_k, is set to be $M_k = 1.2 \times \sum_{i=1}^{n} \lambda_i/m$, $1 \leqslant k \leqslant m$, to make sure that its capacity has an overall excess of 20% to the total call volume of the cells.

Except for otherwise mentioned, we adopt the default setting of parameters mentioned above to run *ACO* and no local search is employed. Note that each value reported in this section is averaged from the results of ten randomly generated problem instances.

5.1.1. Determination of β

Fig. 5 summarizes some results of applying *ACO* to solve the cell assignment problem with different values of β. The other parameters are set as the default values.

From Fig. 5(a), we find that the solution quality of *ACO* using $\beta \geqslant 1$ is much better than that using $\beta = 0$. Fig. 5(b) reveals that *ACO* could find quite appealing solutions within 600 iterations by setting $\beta \geqslant 2$. As can be seen from Fig. 5(c), the CPU time needed by *ACO* using $\beta \geqslant 2$ is no more than 180 s even in solving a large instance like $(n, m) = (250, 12)$.

By setting $\beta = 0$, the heuristic function is disabled and the search session of ants will be directed by the pheromone trails only. The experimental results shown in Fig. 5(a) demonstrate that such a search strategy does not produce good solutions. Without the proper guidance of the greedy heuristic that finds good approximate solutions in the earlier iterations, the ants might waste their efforts in exploring some areas of the solution space that do not contain optimal or even satisfactory approximate solutions. Even though some iteration-best paths explored by pioneering ants are not absolutely informative, the extra pheromone intensity of $\rho\,\Delta\tau$ resulted from global pheromone updates still attracts other ants to follow such that *ACO* tends to converge around some common paths in a short time (see Fig. 5(c)). In fact, the setting of $\beta = 0$ implies that *ACO* is deploying a random heuristic with multiple agents (ants).

To learn more about the performances resulted from different settings of β, we depict the costs of the iteration-best paths with respect to every 50 iterations for a particular problem instance with $(n, m) = (200, 10)$ in Fig. 6. For the comparison purpose, the total number of iterations is set to be 500.

In earlier iterations, $\beta = 3$ or 4 helps find better approximation solutions than $\beta = 0$, 1 or 2 as shown in Fig. 6. In the state transition rule, a larger value of β makes the ants rely more on the greedy heuristic when determining their next moves. Although such a greedy strategy directs ants to good solutions quickly in

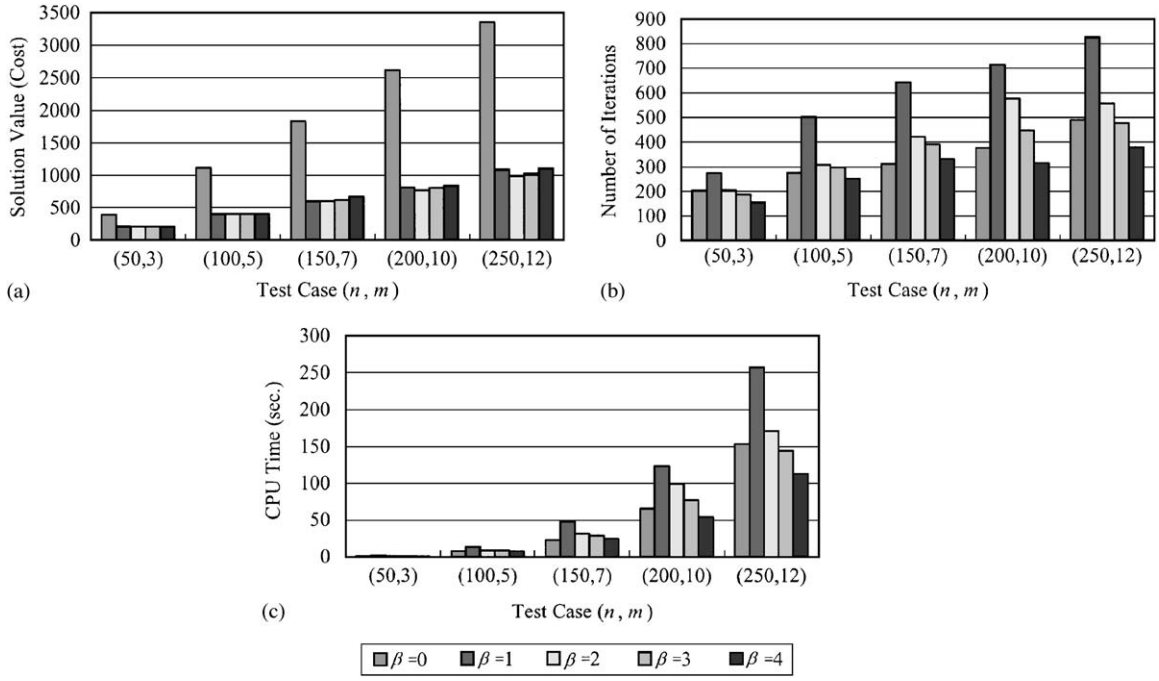

Fig. 5. Results for different values of β, (a) solution values, (b) number of iterations and (c) CPU time elapsed.

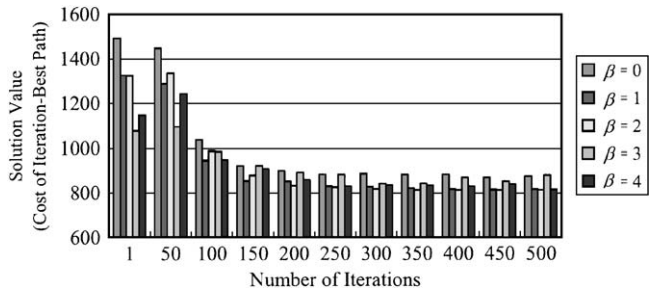

Fig. 6. Solution value with respect to the number of iterations for $\beta = 0, 1, 2, 3$ and 4.

the earlier stage, the accumulated pheromone intensity from global updates on the arcs of some seemly good paths found by the greedy heuristic as far may guide the fellow ants to converge to some common paths, or we may say, some local optimums only. As can be seen in Fig. 6, the solutions found by using $\beta = 1$ or 2 would be better than those by $\beta = 3$ or 4 after 150 iterations. The setting of $\beta = 2$ seems to be the best choice for the trade-off between the guidance of the greedy heuristic and that of the pheromone intensity.

5.1.2. Determination of q_0

In Fig. 7, we show the experimental results of the solution values by setting q_0 as 0, 0.1, 0.3, 0.5, and 0.9 for *ACO*.

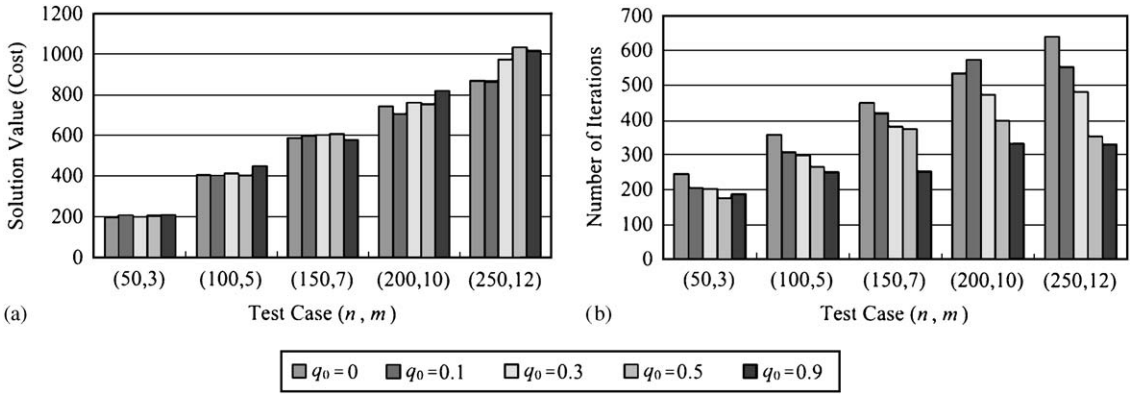

Fig. 7. Results for different values of q_0, (a) solution values and (b) number of iterations.

It can be seen from Fig. 7(a) that the impact imposed by different q_0 values on *ACO* is significant only for large-scale instances, say $n \geqslant 200$ and $m \geqslant 10$. Furthermore, the setting of $q_0 = 0.1$ for *ACO* performs better than other settings in our experiments. Using $q_0 = 0.1$ means that it is more likely for ants to conduct biased exploration than exploitation. That is, it is effective for the ants to depend more on biased exploration in delivering satisfactory solutions. Note that in Dorigo and Gambardella's experiments for solving TSP [14,15], they applied the setting of $q_0 = 0.9$ which suggests ants to count more on exploitation. On the contrary, the setting of $q_0 = 0.9$ yields inferior solution qualities in our test instances. Based upon these observations, we know that it is necessary to tune the parameter values of *ACO* to best fit the characteristics of the problem under study to achieve a better performance.

From Fig. 7(b), we find that the number of iterations needed by *ACO* tends to be large when q_0 is small, while it tends to be small when q_0 is large. It is quite natural that when q_0 is small, the ants have more chances to explore the solution space so that *ACO* needs more iterations. When q_0 is large, the ants have more chances to exploit the good solutions so far and this situation might lead to relatively premature convergence and the number of iterations needed would be relatively small.

5.1.3. *Effects of pheromone update*

To examine the influence different pheromone updating rules might induce, we conduct experiments with four strategies: (1) *Both*: applying both global and local updates, (2) *GloOnly*: applying global update only, (3) *LocOnly*: applying local update only, and (4) *None*: applying no update (i.e., pheromone intensity is not updated). Note that in *Both*, the local pheromone updating rule is the same as the default setting, $\tau = (1 - \varphi)\tau + \varphi\tau_0$, while in *LocOnly*, the local pheromone updating rule is set to be $\tau = (1 - \varphi)\tau + \varphi\,\Delta\tau$ where $\Delta\tau = n/Q^+$. (If we apply the same local update rule as in *Both* for *LocOnly*, the pheromone intensity on all arcs would always be τ_0.) Fig. 8 summarizes the results of some test instances for the four strategies.

Fig. 8(a) demonstrates that *None* and *LocOnly* report the worst solution qualities. *None* keeps the intensities of pheromone trails at a constant (initial) level, i.e., τ_0, which disables the possible influences induced by pheromone trails. Thus, it becomes a pure greedy heuristic with which many ants search the solution space in a random way. The heuristic function leads the ants to converge to only local optimums rapidly (see Fig. 8(b)). With *LocOnly*, the pheromone intensities between before the pheromone decay (τ) and after ($(1 - \varphi)\tau + \varphi\tau$) are so close that the significance of the heuristic value dominates that of

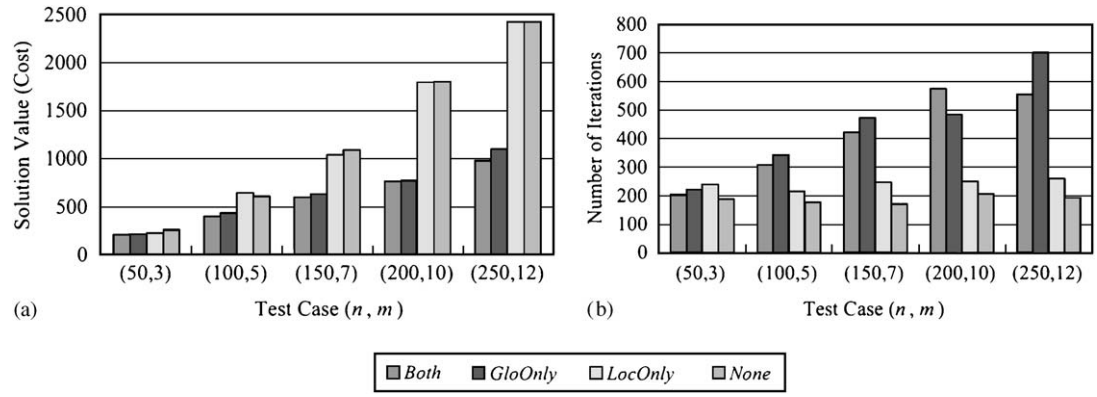

Fig. 8. Results for different pheromone updating strategies, (a) solution values and (b) number of iterations.

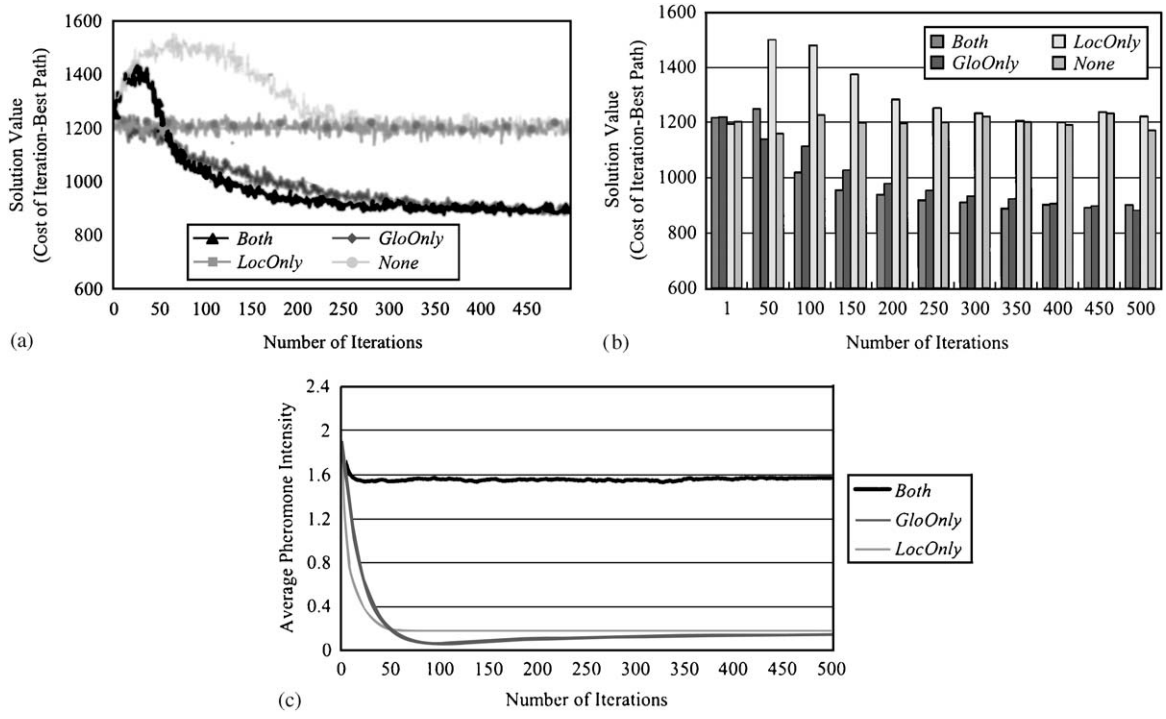

Fig. 9. Results for different settings of pheromone updates. (a) Solution value with respect to number of iterations, (b) solution values with respect to every 50 iterations and (c) average pheromone intensity on the solution path with respect to number of iterations.

pheromone intensity. With *GloOnly*, the solution quality of *ACO* is improved. Furthermore, with *Both*, *ACO* achieves the best solution quality in our experiments.

Fig. 9(a) illustrates the iteration-best costs by using these four pheromone updating strategies with respect to the number of iterations for a particular problem instance with $(n, m) = (200, 10)$. Fig. 9(b)

summarizes the solution values every 50 iterations for better readability. The stopping criterion is set a limit of 500 iterations. The effectiveness of both global and local pheromone updates could be clearly evinced through Fig. 9(a) and (b).

As expected, the curves of *None* and *LocOnly* fluctuate sharply around poor solution values (local optimums). Even in later iterations, their solution qualities are still inferior. Note that the curve of *GloOnly* in Fig. 9(a) tends to drop smoothly. It is the reinforcement due to global pheromone updates that provides reliable guidance for ants to find good solutions. The curves of *Both* or *LocOnly* climb up before the early 30 or 80 iterations, respectively, and they drop afterwards. It is the forgetfulness incurred by local pheromone updates that encourages the ants in *Both* and *LocOnly* to explore more other solution areas in the preliminary iterations so that the iteration-best solutions found during this stage might be worse. In the case of *Both*, after 80 iterations, the joint effect of both reinforcement and forgetfulness in the pheromone updating process leads the ants to find even better solutions than the others.

For the same problem instance, Fig. 9(c) focuses on the average pheromone intensities on the best paths (i.e., $\sum_{\langle i,j\rangle\in\psi}\tau_{ij}/(2n-1)$ where ψ denotes the best path) found by *Both*, *GloOnly* and *LocOnly*, respectively, with respect to the number of iterations. As mentioned before, the significance of heuristic value dominates that of pheromone intensity in *LocOnly*. Thus the arcs in the best path have great chances to be traversed by some ants during the earlier iterations. These arcs would be selected with a high probability by the guidance of the heuristic value, thus their average pheromone intensity would be rapidly decreased to reach a certain minimum value due to pheromone decay. Likewise, in *GloOnly*, the arcs belonging to the best path might also be traversed in the earlier iterations. The pheromone trails on the arcs of the best path evaporate regularly in each iteration and receive additional pheromone whenever ants visit them again (referring to Eq. (10)). When using the default parameters in our experiments, the average pheromone intensity in the best path decreases before the first 100 iterations. As time goes on, the pheromone intensities on the arcs that are seldom visited by ants evaporate only in such a way that they decrease faster than those in the best path. Thus, ants gradually move back and forth along the best path and the average pheromone intensity in the best path starts to increase slightly after iteration 100. In *Both*, ants rapidly concentrate upon the best path with a large possibility and the average pheromone intensity, around 1.57 in this case, is a compromised result among the evaporation and reinforcement from global updates as well as the decay via local updates.

5.1.4. Effects of evaporation in global pheromone update

To demonstrate the effectiveness of the evaporation process in *ACO*, we compare the results of performing (1) *Eva*: evaporation on all arcs (referring to Eq. (10)), and (2) *None*: no evaporation at all (referring to the equation $\tau = \tau + \rho\,\Delta\tau$). Note that the local pheromone update is performed for both cases.

Fig. 10(a) clearly suggests that for most cases *Eva* produces better solutions than *None*. Note that both *Eva* and *None* emphasize the significance of the iteration-best paths by adding some extra pheromone ($\rho\Delta\tau$) on the corresponding arcs, while the evaporation of *Eva* degrades the pheromone intensities for all arcs in each iteration. Let us observe the difference of the pheromone intensities on the arcs between those in the iteration-best paths and those not. Such a difference in *None* would be larger than that in *Eva*. Thus, the ants in *None* tends to converge to some common paths earlier. Fig. 10(b) also evinces such a premature phenomenon of *None* by showing that the number of iterations needed in *None* is smaller than that in *Eva*.

From Fig. 11, which illustrates the solution values of the iteration-best paths with respect to the number of iterations for a particular problem instance with $(n, m) = (200, 10)$ by running 500 iterations,

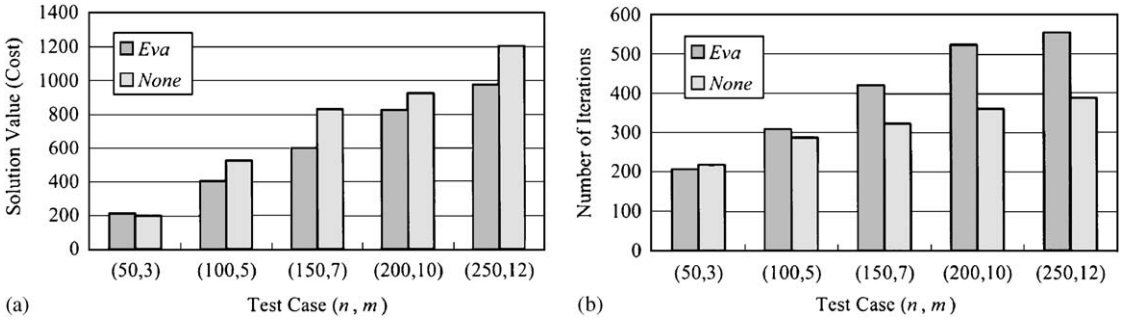

Fig. 10. Results for different modes of pheromone evaporation, (a) solution values and (b) number of iterations.

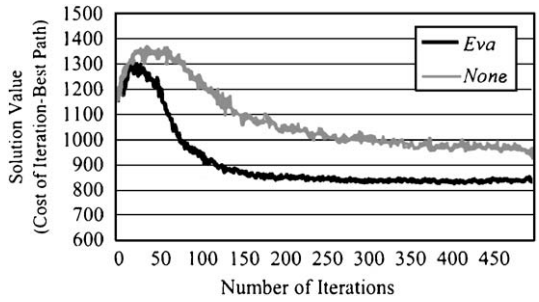

Fig. 11. Solution values for different evaporation modes.

we find that *Eva* leads the ants to better solutions. Both of the curves of *Eva* and *None* climb up in earlier iterations and drop afterward. We might say that evaporation prevents the pheromone trails from accumulating knowledge too fast or forgets bad decisions made in earlier stages of the search process. Without such a proper forgetfulness process on the learned experience, the extra pheromone added by global updates will direct the ants to follow previous experiences without considering new alternatives. This surely increases the risk of being trapped in the situation of stagnation. Note that both curves still fluctuate to a certain degree in the later iterations. This means that during the later iterations observed, the ants would still keep traveling on some alternative paths to search for better solutions.

To determine a proper value for the evaporation coefficient ρ, we test various values and summarize the results in Fig. 12. In most of the test cases, $\rho = 0.05$ outperforms other values.

5.1.5. *Effects of decay in local pheromone update*

Pheromone decay in the local pheromone update process is to reduce the risk that an ant might mostly follow the same path traversed by its predecessors. Whenever an ant moves along an arc, decay occurs on the traversed arc. Fig. 13 compares the results of employing the pheromone decay in the local pheromone update (Dec) (referring to Eq. (11)), and the results of ignoring the local pheromone update ($None$). Note that in both cases, global pheromone updates are adopted.

It can be easily seen from Fig. 13(a) that we have better solution qualities by using *Dec*. The pheromone decay reduces the pheromone intensity on the just visited arcs and suggests ants to explore other arcs.

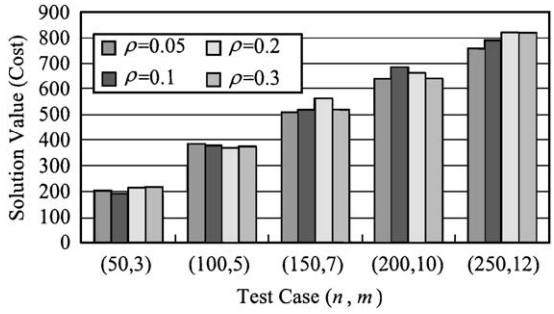

Fig. 12. Solution values for various settings of ρ.

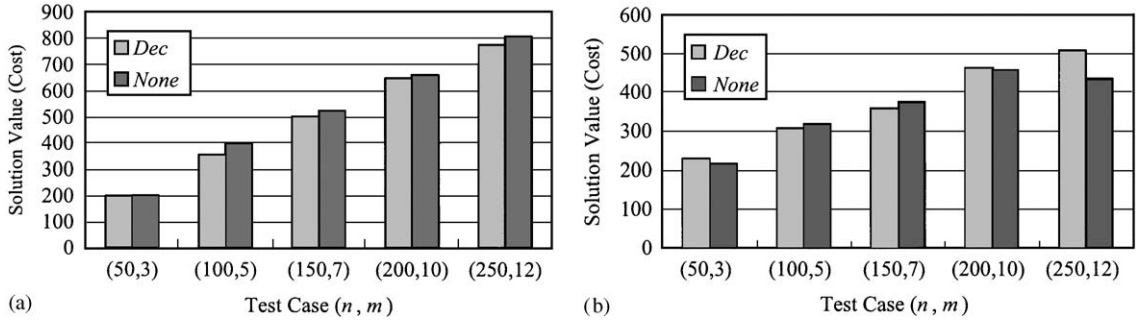

Fig. 13. Solution values and numbers of iterations for various pheromone decay settings, (a) solution values and (b) number of iterations.

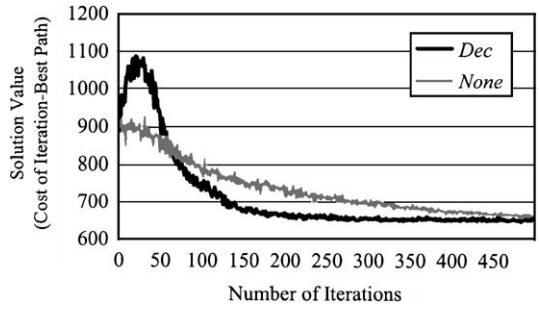

Fig. 14. Solution values for different settings of pheromone decay.

It is also reasonable for *Dec* to run with more iterations than *None*, as shown in Fig. 13(b), before the stopping criterion is satisfied.

Fig. 14 shows the costs of the iteration-best paths for *Dec* and *None* with respect to the number of iterations needed for solving a particular instance with $(n, m) = (200, 10)$ by running *ACO* for 500 iterations. Let us focus on the difference of the pheromone intensities on the arcs between those belonging to the iteration-best paths and those do not. Such a difference in *None* would be larger than that in *Dec*.

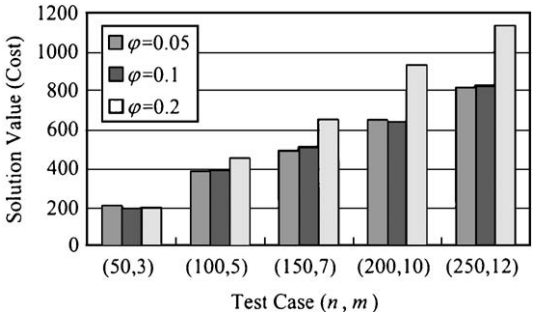

Fig. 15. Solution values for various values of φ.

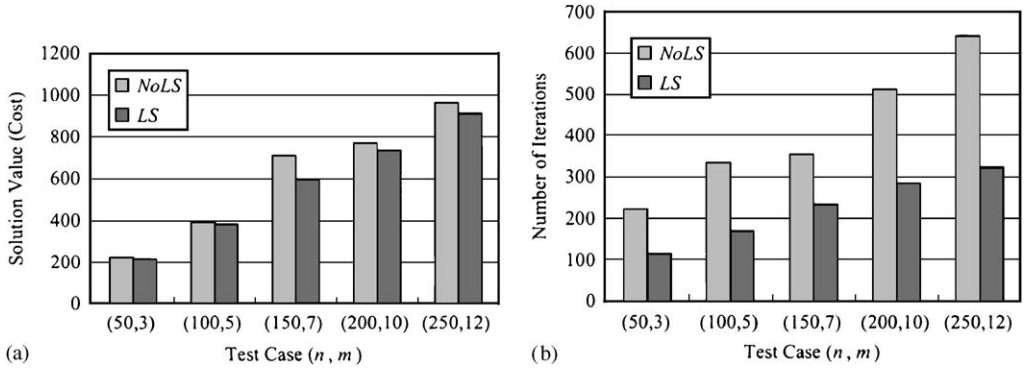

Fig. 16. Effects of local search, (a) solution values and (b) number of iterations.

Thus in *None*, the ants tend to follow the guidance of the pheromone trails on those iteration-best paths found by their predecessors. This makes *None* more likely to be trapped into a stagnation situation.

Fig. 15 shows the solution values derived by using different values of the pheromone decay coefficient φ. Among most of the test cases, the setting of $\varphi = 0.05$ provides more satisfactory solutions than other settings.

5.1.6. *Effects of local search*

The local search is applied whenever a new global best solution is found. The effect of the local search in *ACO* can be justified from Fig. 16. Fig. 16(a) indicates that we see that the solution values obtained by applying the local search (LS) are better than those obtained without local search (*NoLS*). Fig. 16(b) also discloses that *ACO* with local search would be more efficient in convergence speed as compared with *ACO* without local search. Since the application of local search might improve the best cost found so far from Q^+ to be Q'^+ ($Q'^+ \leqslant Q^+$), the pheromone deposited in global update $\Delta\tau_{LS}(=n/Q'^+)$ would be no less than $\Delta\tau_{NoLS}(=n/Q^+)$. That is, more pheromone might be deposited on the arcs of the paths that are explored by adopting local search to guide the following ants. Such guidance is informative because some of these arcs might have great chances to be the ones in the best path. Subsequently, the time needed in *LS* might be reduced. That is, the local search helps the ants not only to get better solutions but also to reduce the time for convergence in our experiments.

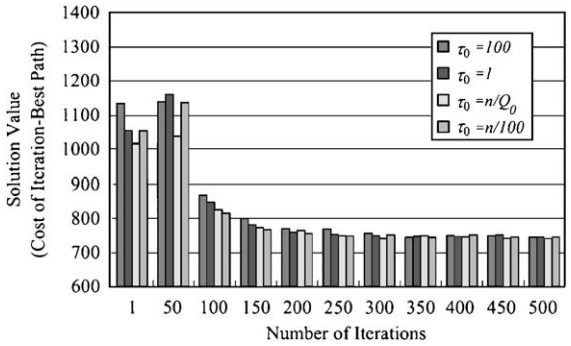

Fig. 17. Results for different settings of initial pheromone.

5.1.7. Determination of τ_0

We define the value of the initial pheromone τ_0 to be $n/100$, which depends only upon the number of cells. As matter of fact, this value is not a critical factor to the quality of the solutions reported by *ACO* in our experiments. This can be observed from Fig. 17, which illustrates the solution values by adopting various settings of τ_0, including $\tau_0 = 100$, 1, n/Q_0 or $n/100$, with respect to the number of iterations for a particular instance with $(n, m) = (200, 10)$, where Q_0 is the cost of an approximate solution produced by the heuristic in Merchant and Sengupta [4]. Our setting of τ_0 as $n/100$ instead of $n(Q_0)^{-1}$ also ensures that *ACO* alone can handle the problem and there is no need to apply some heuristic to start with.

5.1.8. Determination of u

It is intuitive that more ants participating in the search process cooperatively might provide a synergetic effect by exchanging their individual experiences via pheromone trails. To determine an appropriate number of ants in *ACO*, we compare the performances resulted from different numbers of ants in Fig. 18. Fig. 18(a) indicates that when the problem instances become large, the solution values reported by $u = 16$ or 20 are slightly better than those by $u = 1$, 4, or 10. It appears that employing more ants could reinforce the cooperation among ants. Fig. 18(b) shows that the numbers of iterations need by various numbers of ants are similar (except for $u = 4$). However, using a similar number of iterations does not imply a similar elapsed time in this experiment. In fact, as shown in Fig. 18(c), it takes more execution time when more ants are employed. We set $u = 16$ because this setting achieves a reliable performance with respect to solution quality in a reasonable execution time.

5.2. Comparison with other approaches

In this section, we generate the test instances by adopting the scheme suggested in Merchant and Sengupta [4] and Pierre and Houeto [8,9]. We summarize their instance generation scheme as follows. Cabling cost between cell i and switch k, c_{ik}, is set to be proportional to the geometric distance between them. The call rate λ_i of cell i follows a Gamma distribution with mean one and coefficient of variation 0.25. The call duration inside the cells is exponentially distributed with mean one. If cell j has t neighbors, we divide [0, 1] into $t + 1$ intervals by selecting t random numbers from a uniform distribution between 0 and 1. At the end of the service period in cell j, the call could be either transferred to the ith neighbor, $1 \leqslant i \leqslant t$, with a handoff probability r_{ij} equal to the length of the ith interval, or ended with a probability

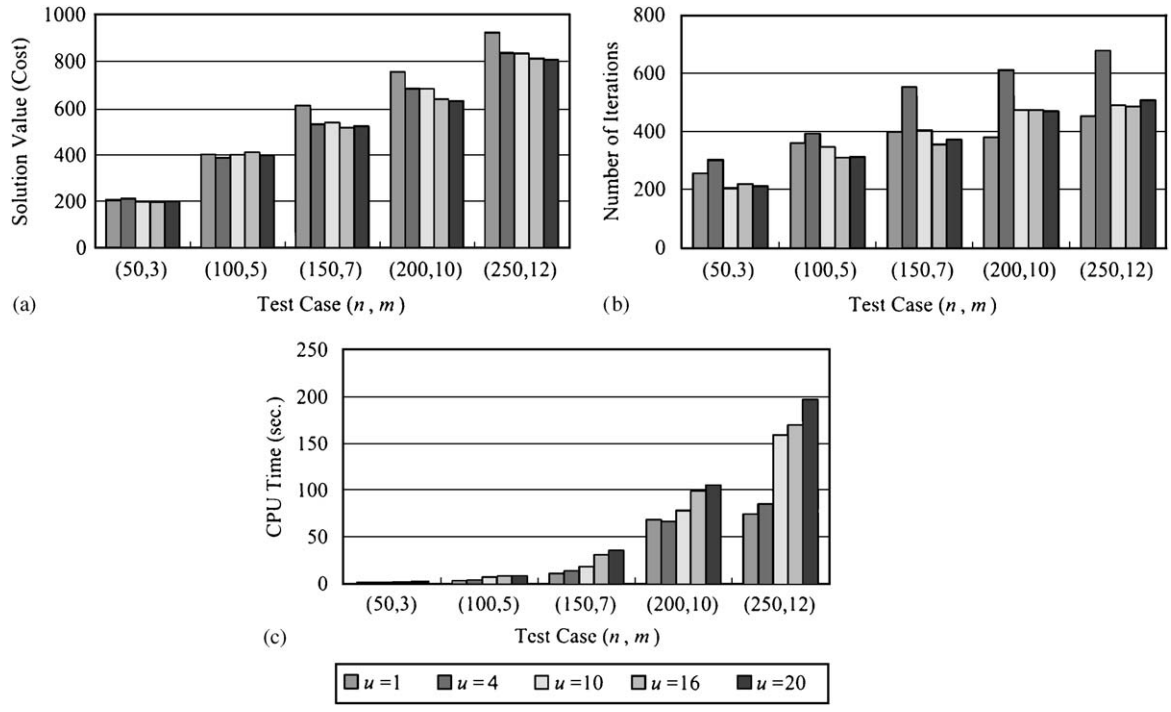

Fig. 18. Results for different numbers of ants, (a) solution values, (b) number of iterations and (c) CPU time elapsed.

equal to the length of the $(t+1)$th interval. A Jackson network [32] is applied to find a consistent set of call volume and handoff rate. The incoming rate α_i in cell i are computed from the following system:

$$\alpha_i - \sum_{j=1}^{n} \alpha_j r_{ji} = \lambda_i, \quad 1 \leqslant i \leqslant n.$$

The handoff rate h_{ij} is defined by $h_{ij} = \lambda_i r_{ij}$. All switches have the same capacity as

$$M_k = (1 + K/100) \times \sum_{i=1}^{n} \lambda_i / m, \quad 1 \leqslant k \leqslant m,$$

where K is a random number within [10, 50] to make sure a global excess of 10–50% of the switches' capacity compared with cells' volume of call.

To compare the performance of *ACO* with other approaches mentioned in this paper, we implement heuristics, including *H* [4], *H*-II and *H*-IV [6], and metaheuristics, including *SA* [7], TS [8,9] and *MA* [10], which incorporates *TS* as local search [10]. The parameters involved in the metaheuristics have been tested in advance to get their better performances. Note that *ACO* is equipped with the local search procedure.

First of all, we would like to know the error ratios of the solutions given by the approaches with respect to the optimal solutions. Thus, we implement a straightforward enumerative algorithm to find optimal solutions (*OPT*). Due to time limit, only small-size problem instances are considered. The problem sizes

Table 1
Comparison between heuristics/metaheuristics and *OPT*

(n, m)	Error ratio (%)							CPU time (s)					
	H	*H*-II	*H*-IV	*SA*	*TS*	*MA*	*ACO*	*H*/*H*-II/*H*-IV	*SA*	*Tabu*	*MA*	*ACO*	*OPT*
(5, 2)	0.27	0.68	0.68	0.00	0.00	0.00	0.01	< 0.01	1.66	0.07	0.75	0.02	<0.01
(5, 3)	1.23	1.28	1.28	0.14	0.14	0.14	0.00	<0.01	0.29	0.01	0.20	0.02	<0.01
(8, 2)	1.41	1.40	1.31	0.11	0.11	0.10	0.01	<0.01	0.31	0.01	0.61	0.02	0.02
(8, 3)	1.66	1.62	1.62	0.47	0.47	0.46	0.51	<0.01	0.40	0.02	0.38	0.03	0.39
(10, 2)	4.21	5.41	4.21	6.61	5.82	6.03	5.83	<0.01	2.00	0.10	0.43	0.03	0.11
(10, 3)	1.89	3.80	3.44	4.71	3.55	2.09	3.47	<0.01	0.49	0.02	0.32	0.04	5.13
(18, 2)	2.13	1.32	1.69	0.07	0.07	0.07	0.01	<0.01	2.60	0.11	2.43	0.08	57.24
(20, 2)	1.54	1.28	1.24	0.03	0.03	0.13	0.00	<0.01	1.53	0.11	1.03	0.09	278.53
(25, 2)	1.91	1.62	1.74	2.66	1.02	1.08	0.52	<0.01	1.79	0.53	3.43	0.19	12713.32

are $n = 5, 8, 10, 18, 20$ or 25, and $m = 2$ or 3. Ten problem instances are generated for each pair of n and m and the results are averaged. Table 1 summarizes the error ratios and the elapsed CPU time.

As can be seen from Table 1, these heuristics or metaheuristics can produce quite attractive solutions. On some occasions, even optimal solutions could be found. Heuristics H, H-II and H-IV are much faster than those metaheuristics. However, in most of our test cases, the solution qualities obtained by metaheuristics are better than those by the three heuristics. For some smaller problem sizes, *TS* or *MA* give better solutions than *ACO*. But, for most of the test cases, *ACO* obtains slightly better results. Furthermore *ACO* has a shorter elapsed time as compared with other metaheuristics. Note that *OPT* is only a straightforward implementation to obtain the optimal solutions for the comparison purpose. Therefore, the CPU time elapsed by *OPT* is relatively huge.

For large problem instances, we set the sizes to be $n = 25, 50, 100, 150, 200$ or 250 and $m = 2, 3, 5, 7, 10$ or 15. Besides, two larger problem instances with $n = 350$ or 500, and $m = 20$ are also tested. Thirty problem instances are generated for each pair of n and m. To have a concise comparison between the solutions obtained through different approaches, we derive the relative deviations of solution values using the solution reported by *ACO* as the baseline. Table 2 summarizes the deviations in percentages. The CPU time needed by these approaches are averaged and summarized in Table 3.

From Table 2, we realize that in most of the test cases, *ACO* reports better solutions than the other approaches, whereas, in a few cases, *TS* gets better solutions. When the size of the problem is larger, the effectiveness of *ACO* becomes more significant. The solution qulity of *MA* approaches to that of *ACO*. We may say that these metaheuristics are competitive to each other. In general, the solutions obtained from metaheuristics are better than those from simple heuristics when the size of the problem instances becomes large, while for some cases that the number of switches is small, heuristics might have appealing results. The performance of *SA* is also pretty good for some small n or m (say $n = 25, 50$ or $m = 2, 3$).

The results of execution time needed by these approaches are reported in Table 3. It reveals that simple heuristics can find out solutions in a very short time. This phenomenon is reasonable because metaheuristics need more iterations before they satisfy the stopping criterion. On the other hand, metaheuristics usually produce better solutions. For the most time-consuming case with $n = 500$ and $m = 20$, SA, TS and *ACO* take about 36, 32 and 25 min, respectively, on an average. Although H(H-II and H-IV) takes only 3 s (0.3 and 0.3, respectively), the solution is 19.43% (14.14% and 12.63%, respectively) worse than

Table 2
Relative deviation for solution values by different heuristics with respect to *ACO*

(n, m)	(*H-ACO*)/*ACO* ×100 (%)	(*H*-II-*ACO*)/*ACO* ×100 (%)	(*H*-IV-*ACO*)/*ACO* ×100 (%)	(*SA-ACO*)/*ACO* ×100 (%)	(*TS-ACO*)/*ACO* ×100 (%)	(*MA-ACO*)/*ACO* ×100 (%)
(25, 2)	1.67	3.43	2.42	2.58	3.44	3.04
(25, 3)	3.24	7.53	6.05	1.23	0.06	3.99
(25, 7)	1.86	9.08	8.37	1.19	−0.03	0.65
(25, 10)	2.48	8.92	7.05	0.80	2.78	0.63
(25, 15)	2.92	13.82	12.97	0.42	4.09	1.22
(50, 2)	1.91	1.93	2.78	0.43	−1.54	1.05
(50, 3)	6.24	6.61	4.67	1.03	−0.91	2.82
(50, 5)	8.17	7.52	7.04	1.50	2.36	3.98
(50, 7)	8.34	9.17	6.23	1.37	7.88	1.64
(50, 10)	8.94	10.71	10.46	1.82	2.31	1.99
(50, 15)	7.62	10.59	9.08	2.77	0.66	0.99
(100, 2)	6.83	3.29	2.08	1.53	0.90	1.53
(100, 3)	6.37	4.53	6.69	1.32	1.80	2.67
(100, 5)	6.02	6.80	7.74	3.62	5.97	4.16
(100, 7)	15.04	8.62	9.49	4.14	−0.53	0.64
(100, 10)	11.30	8.94	8.37	5.52	5.74	4.58
(100, 15)	9.88	11.98	10.08	6.88	9.82	4.50
(150, 2)	13.81	3.69	2.29	0.64	0.94	1.35
(150, 3)	14.45	2.35	1.88	1.55	−0.91	3.14
(150, 5)	14.99	6.57	5.05	7.25	11.20	0.94
(150, 7)	16.85	8.97	8.81	3.41	2.78	1.63
(150, 10)	14.85	13.26	11.58	5.40	3.77	1.60
(150, 15)	13.34	11.86	11.67	7.46	9.88	3.12
(200, 2)	6.45	1.47	0.84	0.62	0.77	0.34
(200, 3)	9.69	4.95	2.77	1.21	0.99	0.06
(200, 5)	10.32	5.51	4.52	2.57	0.84	3.46
(200, 7)	10.49	6.88	7.41	4.13	9.52	3.16
(200, 10)	19.66	11.96	9.69	6.54	8.27	3.87
(200, 15)	22.64	14.12	12.57	7.37	10.92	4.89
(250, 2)	0.75	2.47	1.53	1.14	4.13	1.71
(250, 3)	1.56	2.83	0.48	3.13	3.06	3.48
(250, 5)	12.43	8.12	4.48	2.80	4.34	0.24
(250, 7)	19.64	6.71	5.25	1.40	6.94	1.32
(250, 10)	11.68	6.98	6.95	3.77	6.52	3.21
(250, 15)	15.71	13.52	9.85	6.75	4.68	2.98
(350, 20)	18.71	12.82	12.66	8.58	4.18	3.47
(500, 20)	19.43	14.14	12.63	10.77	4.29	4.11

Table 3
CPU time (s) needed by different heuristics

(n, m)	*H*	*H*-II	*H*-IV	*SA*	*TS*	*MA*	*ACO*
(25, 2)	<0.1	<0.1	<0.1	1.8	0.5	3.4	0.2
(25, 3)	<0.1	<0.1	<0.1	0.9	0.2	3.3	0.2
(25, 7)	<0.1	<0.1	<0.1	0.8	0.6	2.8	0.4
(25, 10)	<0.1	<0.1	<0.1	1.1	0.7	2.5	0.7
(25, 15)	<0.1	<0.1	<0.1	2.3	1.2	3.7	1.1
(50, 2)	<0.1	<0.1	<0.1	1.3	0.7	20.4	0.5
(50, 3)	<0.1	<0.1	<0.1	1.3	1.7	23.9	0.6
(50, 5)	<0.1	<0.1	<0.1	1.8	1.7	17.3	1.2
(50, 7)	<0.1	<0.1	<0.1	3.1	4.2	17.7	2.1
(50, 10)	<0.1	<0.1	<0.1	2.9	3.6	20.5	3.3
(50, 15)	<0.1	<0.1	<0.1	3.6	5.0	16.5	4.7
(100, 2)	<0.1	<0.1	<0.1	3.7	4.8	89.3	2.0
(100, 3)	<0.1	<0.1	<0.1	6.0	5.8	94.1	4.1
(100, 5)	<0.1	<0.1	<0.1	8.7	5.6	102.2	5.6
(100, 7)	<0.1	<0.1	<0.1	14.4	6.6	105.9	7.7
(100, 10)	<0.1	<0.1	<0.1	17.4	8.7	104.5	7.3
(100, 15)	0.1	<0.1	<0.1	29.9	14.7	144.7	16.4
(150, 2)	<0.1	<0.1	<0.1	8.8	10.8	662.7	3.2
(150, 3)	<0.1	<0.1	<0.1	14.2	11.6	785.0	5.9
(150, 5)	0.1	< 0.1	<0.1	28.4	15.2	693.7	12.8
(150, 7)	0.1	<0.1	<0.1	37.6	14.6	679.5	18.8
(150, 10)	0.2	<0.1	<0.1	40.5	15.7	742.9	33.0
(150, 15)	0.1	<0.1	<0.1	50.6	42.7	697.0	45.0
(200, 2)	<0.1	<0.1	<0.1	14.5	18.3	1834.5	9.7
(200, 3)	<0.1	<0.1	<0.1	16.7	27.9	2128.1	13.2
(200, 5)	0.1	<0.1	<0.1	22.9	19.8	2561.3	19.7
(200, 7)	0.1	<0.1	<0.1	30.2	33.6	2270.4	34.5
(200, 10)	0.2	<0.1	<0.1	39.7	41.7	2735.2	62.3
(200, 15)	0.2	0.1	<0.1	<0.1	<0.1	2800.7	88.6
(250, 2)	0.1	<0.1	<0.1	87.5	22.3	6729.4	13.2
(250, 3)	0.1	<0.1	<0.1	100.5	41.8	7416.7	19.6
(250, 5)	0.1	<0.1	<0.1	105.6	41.7	7603.9	42.2
(250, 7)	0.2	0.1	0.1	221.7	49.1	7528.0	62.0
(250, 10)	0.2	0.1	0.1	188.3	112.2	7329.2	90.8
(250, 15)	0.3	0.2	0.1	309.2	236.0	7910.7	178.4
(350, 20)	1.2	0.1	0.1	1072.8	943.4	86155.8	735.5
(500, 20)	3.0	0.3	0.3	2152.7	1884.6	119328.7	1495.3

that found by *ACO*. Therefore it is informative for the designers of PCS networks to apply metaheuristics such as SA, TS or *ACO* to resolve the cell assignment problem if the cost minimization is the most critical concern. On the contrary, if a prompt response is demanded, heuristics like H-II or H-IV are quite attractive to offer feasible solutions. Although the solution values obtained by *MA* are competitive to *ACO*, it takes a much longer execution time (about 33 h for the problem size of $(n, m) = (500, 20)$) than *ACO* (about 25 min). *MA* might be impractical for real-world applications. We would suggest that *ACO* is an effective approach to delivering solutions with satisfactory qualities in a reasonable time, even though the statistics of our experiments does not imply the absolute superiority over the other approaches for all of the test cases.

6. Concluding remarks

The problem of cell-to-switch assignment is essential to the development of PCS or global communication services. In this paper, we have developed an *ACO* algorithm to solve this problem. We model the problem in a form of the matching problem in a weighted directed bipartite graph so that our artificial ants can construct their paths on the graph. Such an abstract structure might be helpful for ACO designers to model their own problems. We have also conducted experiments to capture the behavior of ants in problem optimization and empirically study the performances of ACO and other approaches with large-scale problem instances.

Numerical results of the experiments have demonstrated the effectiveness of our *ACO* algorithm in coping with the cell assignment problem. Although *ACO* takes a longer time than the three heuristics H, H-II and H-IV, it could find much better approximate solutions. In fact, the time needed by *ACO* is also reasonable when compared with those needed by exact algorithms. When compared with other metaheuristics, *ACO* is also effective and efficient with respect to both solution quality and execution time. The concern of solution quality against temporal issue is a trade-off for PCS designers. For the cell assignment problem, *ACO* is a promising approach for practical implementations. It is also easy to tailor *ACO* to resolve some extensions of the cell assignment problem such as the *on-line* version where the cells or switches would grow or shrink due to environment change or communications traffic, or the *constrained* version where precedence or priority relationships among cells and switches might be considered, just to name a few. At the same time the paper was published, an independent study by Fournier and Pierre [33] hybridized the *k*-opt local optimization technique with the conventional ACO meta-heuristic to deal with the problem considered.

Since the decision about moving from a cell to a switch and that about moving from a switch to a cell are intrinsically different, we employ two heuristic functions to accommodate their effectiveness respectively. This idea is different from most of the traditional ant algorithms. It is worthy of generalizing this idea of our ACO algorithms for solving combinatorial optimization problems that could be structured by hybrid decision criteria. The parameter setting used in our ACO algorithm is quite different from those used in Dorigo et al. [14] and Dorigo and Gambardella [15]. It means that ACO needs to be tuned, in accordance with the characteristics of the studied problem, to establish the appropriate search strategy. When tuning the parameters, we examine the behavior of ACO in optimizing the problem. These experiences might encourage us to use the ACO algorithms to deal with other combinatorial optimization problems.

Acknowledgements

The authors would like to express their appreciation to the anonymous referees and the Editor-in-Chief for their constructive comments which make the paper more comprehensive. This research was partially supported by the National Science Council of the ROC under Grants NSC-90-2213-E-130-001 and NSC-91-2416-H-260-001.

References

[1] Cheng M, Rajagopalan S, Chang LF, Pollini GP, Barton M. PCS mobility support over fixed ATM networks. IEEE Communication Magazine 1997;35:82–92.
[2] Akyildiz IF, McNair J, Ho J, Uzunalioglu H, Wang W. Mobility management in current and future communication networks. IEEE Network Magazine, 1998.
[3] Yacoub MD. Foundations of mobile radio engineering. Boca Raton, FL: CRC Press; 1993.
[4] Merchant A, Sengupta B. Assignment of cells to switches in PCS network. IEEE/ACM Transactions on Networking 1995;3(5):521–6.
[5] Garey MR, Johnson DS. Computer and intractability: a guide to the theory of $\mathcal{NP}$-completeness. New York: W.H. Freeman; 1979.
[6] Bhattacharjee PS, Saha D, Mukherjee A. Heuristics for assignment of cells to switches in a PCSN: a comparative study. ICPWC'99, 1999. p. 331–4.
[7] Demirkol I, Ersoy C, Caglayan MU, Delic H. Location area planning in cellular networks using simulated annealing. INFOCOM, 2001. p. 13–20.
[8] Pierre S, Houéto F. A tabu-search approach for assigning cells to switches in cellular mobile networks. Computer Communications 2002;25(5):464–77.
[9] Pierre S, Houéto F. Assigning cells to switches in cellular mobile networks using taboo search. IEEE Transactions on Systems, Man, And Cybernetics—Part B 2002;32(3):351–6.
[10] Quintero A, Pierre S. Evolutionary approach to optimize the assignment of cells to switches in personal communication networks. Computer Communications 2003;26(9):927–38.
[11] Menon S, Gupa R. Assigning cells to switches in cellular networks by incorporating a pricing mechanism into simulated annealing. IEEE Transactions on Systems, Man, and Cybernetics—Part B 2004;34(1):558–65.
[12] Dorigo M, Maniezzo V, Colorni A. Positive Feedback as a Search Strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.
[13] Dorigo M. Optimization, learning and natural algorithms. Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992, (in Italian).
[14] Dorigo M, Maniezzo V, Colorni A. The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics — Part B 1996;26(1):29–41.
[15] Dorigo M, Gambardella LM. Ant colonies for the traveling salesman problem. BioSystems 1997;43:73–81.
[16] Costa D, Hertz A. Ants can color graphs. Journal of the Operational Research Society 1997;48:295–305.
[17] Maniezzo V, Colorni A. The ant system applied to the quadratic assignment problem. IEEE Transactions on Knowledge and Data Engineering 1999;11(5):2063–70.
[18] Shyu SJ, Yin PY, Lin BMT, Haouari M. Ant-Tree: an ant colony optimization approach to the generalized minimum spanning tree problem. Journal of Experimental and Theoretical Artificial Intelligence 2003;15(1):103–12.
[19] Gambardella LM, Dorgio M. Ant colony system hybridized with a new local search for the sequential ordering problem. INFORMS Journal on Computing 2000;3:237–55.
[20] Di Caro G, Dorigo M. Mobile agents for adaptive routing. Proceedings of the 31st Hawaii International Conference on System, IEEE Computer Society Press, Los Alamitos, CA, 1998. p. 74–83.
[21] Bullnheimer B, Hartl RF, Strauss C. Applying the ant system to the vehicle routing problem. In: Voß S, Martello S, Osman IH, Roucairol C, editors. Metaheuristics: advances and trends in local search paradigms for optimization. Boston, MA: Kluwer Academic Publishers; 1999. p. 285–96.

[22] Gambardella LM, Taillard ED, Agazzi G. MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. London, UK: McGraw-Hill; 1999. p. 63–76.
[23] Jayaraman VK, Kulkarni BD, Karale S, Shelokar P. Ant colony framework for optimal design and scheduling of batch plants. Computers and Chemical Engineering 2000;24:1901–12.
[24] McMullen PR. An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives. Artificial Intelligence in Engineering 2001;15(3):309–17.
[25] T'kindt V, Monmarché N, Tercinet F, Laügt D. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. European Journal of Operational Research 2002;142(2):250–7.
[26] Merkle D, Middendorf M, Schmeck H. Ant colony optimization for resource-constrained project scheduling. IEEE Transactions on Evolutionary Computation 2002;6(4):333–46.
[27] Di Caro G, Dorigo M. AntNet: distributed stigmergetic control for communications networks. Journal of Artificial Intelligence Research 1998;9:317–65.
[28] Maniezzo V, Carbonaro A. An ANTS heuristics for the frequency assignment problem. Future Generation Computer Systems 2000;16:927–35.
[29] Schoonderwoerd R, Holland O, Bruten J, Rothkrantz L. Ant-based load balancing in telecommunications networks. Adaptive Behavior 1997;5(2):169–207.
[30] Varela GN, Sinclair MC. Ant colony optimisation for virtual-wavelength-path routing and wavelength allocation. Proceedings of the Congress on Evolutionary Computation (CEC'99), Washington DC, USA, July 1999.
[31] Dorigo M, Di Caro G, Gambardella LM. Ant algorithm for discrete optimization. Artificial Life 1999;5:137–72.
[32] Kleinrock L. Queueing systems I: theory. New York: Wiley; 1975.
[33] Fournier JRL, Pierre S. Assigning cells to switches in mobile networks using an ant colony optimization heuristic. Computer Communications 2005; 28(1):65–73.